数据预处理

编码（Encoding）实践工作册

参考答案与解析

仅供教师参考 | 请勿提前分发给学生

版本：1.0 | 2026年3月

第二章 标签编码（Label Encoding）答案

✅ 练习 2-1：对 color 列进行标签编码

```
from sklearn.preprocessing import LabelEncoder


le_color = LabelEncoder()
data['color_label'] = le_color.fit_transform(data['color'])


print('编码结果：')
print(data[['color', 'color_label']])
print()
print('类别映射关系：')
for cls, idx in zip(le_color.classes_, range(len(le_color.classes_))):
    print(f'  {cls} -> {idx}')
```

预期输出：

```
编码结果：
  color  color_label
0    红            2
1    蓝            1
2    绿            0
3    红            2
4    蓝            1
...

类别映射关系：
  绿 -> 0
  蓝 -> 1
  红 -> 2
```

> 📝 **解析：** LabelEncoder 按照字符串的字典顺序（拼音顺序）进行编码，所以 绿(lǜ) < 蓝(lán) < 红(hóng) 并不一定严格按拼音，而是按 Unicode 编码排序。具体顺序可通过 le_color.classes_ 确认。

✅ 练习 2-2：反向解码

```
# 反向解码：将数字还原为原始类别
encoded_values = data['color_label'].values
decoded_values = le_color.inverse_transform(encoded_values)
```

```
print('解码结果：')
for enc, dec in zip(encoded_values[:5], decoded_values[:5]):
    print(f'  {enc} -> {dec}')


# 验证解码是否与原始数据一致
print(f'\n解码是否完全一致: {all(decoded_values == data["color"])}')
```

预期输出：

```
解码结果：
  2 -> 红
  1 -> 蓝
  0 -> 绿
  2 -> 红
  1 -> 蓝

解码是否完全一致: True
```

第三章 独热编码（One-Hot Encoding）答案

✅ 练习 3-1：对 color 列进行独热编码

方法一：pandas get_dummies

```
# 方法一：pandas
color_dummies = pd.get_dummies(data['color'], prefix='color')
print('方法一结果：')
print(color_dummies)
```

方法二：scikit-learn OneHotEncoder

```
# 方法二：scikit-learn
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(sparse_output=False, drop=None)
color_ohe = ohe.fit_transform(data[['color']])

print('方法二结果：')
print('特征名:', ohe.get_feature_names_out())
print(color_ohe)
```

预期输出：

```
方法一结果：
   color_红  color_蓝  color_绿
0     True   False   False
1    False    True   False
2    False   False    True
...

方法二结果：
特征名: ['color_红' 'color_蓝' 'color_绿']
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 ...]
```

📝 **解析：** 两种方法的主要区别：1) pandas 默认输出布尔值 (True/False)，scikit-learn 输出数值 (1.0/0.0)；2) pandas 输出 DataFrame，scikit-learn 输出 NumPy 数组；3)

列顺序可能不同。

✅ 练习 3-2：使用 drop='first' 避免多重共线性

```
from sklearn.preprocessing import OneHotEncoder


ohe_drop = OneHotEncoder(sparse_output=False, drop='first')
color_dropped = ohe_drop.fit_transform(data[['color']])


print('使用 drop="first" 后的结果：')
print('特征名:', ohe_drop.get_feature_names_out())
print(color_dropped[:5])
print()
print(f'原始维度: {data["color"].nunique()} 列')
print(f'删除后维度: {color_dropped.shape[1]} 列')
```

预期输出：

```
特征名: ['color_蓝' 'color_绿']   # 删除了第一个类别 '红'
[[0. 0.]    # 红: 两列都为0即代表红
 [1. 0.]    # 蓝
 [0. 1.]    # 绿
 ...]

原始维度: 3 列
删除后维度: 2 列
```

📝 **解析：** 删除一列不会丢失信息，因为被删除的类别可以通过其他列全为0来唯一确定。例如当 color_蓝=0 且 color_绿=0 时，必定是红色。这就是为什么 n 个类别只需要 n-1 个二进制变量即可完全表示。

第四章 序数编码（Ordinal Encoding）答案

✅ 练习 4-1：对客户满意度进行序数编码

```
from sklearn.preprocessing import OrdinalEncoder
import pandas as pd


# 创建包含满意度的数据
satisfaction_data = pd.DataFrame({
    'customer_id': range(1, 11),
    'satisfaction': ['满意', '非常不满意', '一般', '非常满意',
                     '不满意', '满意', '一般', '非常满意',
                     '不满意', '满意']
})


# 定义顺序：非常不满意 < 不满意 < 一般 < 满意 < 非常满意
sat_order = [['非常不满意', '不满意', '一般', '满意', '非常满意']]
oe = OrdinalEncoder(categories=sat_order)
satisfaction_data['sat_encoded'] = oe.fit_transform(
    satisfaction_data[['satisfaction']]
)


print(satisfaction_data)
print()


# 验证顺序
print('顺序验证：')
for cat, val in zip(sat_order[0], range(len(sat_order[0]))):
    print(f'  {cat} -> {val}')
```

预期输出：

```
   customer_id  satisfaction  sat_encoded
0            1          满意          3.0
1            2      非常不满意          0.0
2            3          一般          2.0
3            4        非常满意          4.0
4            5         不满意          1.0
...


顺序验证：
  非常不满意 -> 0
  不满意 -> 1
  一般 -> 2
  满意 -> 3
  非常满意 -> 4
```

☑ 练习 4-2：思考题 — 对 city 使用 OrdinalEncoder 的问题

参考答案：

city 是无序分类变量，城市之间没有自然的大小顺序。如果强行使用 OrdinalEncoder，会引入以下问题：

1. 虚假的序关系：例如编码为 北京=0、上海=1、广州=2、深圳=3，模型会认为“深圳 > 广州 > 上海 > 北京”，但这种大小关系没有实际意义。

2. 对线性模型影响尤其严重：线性回归、逻辑回归等模型会利用数值大小关系进行计算，导致错误的模型学习。

3. 对树模型影响较小：决策树、随机森林等模型仅使用分裂点，不依赖数值大小关系，因此影响相对较小。

4. 正确做法：对 city 应该使用 One-Hot Encoding、Binary Encoding 或 Target Encoding 等不引入序关系的方法。

第五章 目标编码（Target Encoding）答案

✅ 练习 5-1：对 education 列进行目标编码

方法一：手动实现

```
# 手动计算每个学历的平均工资
edu_mean = data.groupby('education')['salary'].mean()
data['edu_target_manual'] = data['education'].map(edu_mean)


print('手动目标编码结果：')
print(data[['education', 'salary', 'edu_target_manual']])
print()
print('各学历平均工资：')
print(edu_mean)
```

方法二：category_encoders

```
import category_encoders as ce


te_edu = ce.TargetEncoder(cols=['education'])
data['edu_target_ce'] = te_edu.fit_transform(
    data['education'], data['salary']
)


print('对比两种方法的结果：')
print(data[['education', 'edu_target_manual', 'edu_target_ce']])
```

预期输出：

```
各学历平均工资：
education
博士    26333.33
学士    13333.33
硕士    21000.00
高中     8000.00


注意：category_encoders 的结果会略有不同，因为使用了贝叶斯平滑
```

> **解析：** 两种方法的结果不完全一致，因为 category_encoders 的 TargetEncoder 默

认使用贝叶斯平滑（将每个类别的均值向全局均值收缩）来减少目标泄露。样本数越少的类别，收缩越明显。

✅ 练习 5-2：目标泄露实验

```
from sklearn.model_selection import train_test_split

# 分割数据
train_data, test_data = train_test_split(
    data, test_size=0.3, random_state=42
)

# 方式一：在全量数据上计算（有泄露风险）
all_mean = data.groupby('city')['salary'].mean()
train_data['city_target_all'] = train_data['city'].map(all_mean)
test_data['city_target_all'] = test_data['city'].map(all_mean)

# 方式二：仅在训练集上计算（正确做法）
train_mean = train_data.groupby('city')['salary'].mean()
train_data['city_target_train'] = train_data['city'].map(train_mean)
test_data['city_target_train'] = test_data['city'].map(train_mean)

print('方式一（全量数据）- 测试集：')
print(test_data[['city', 'salary', 'city_target_all']])
print()
print('方式二（仅训练集）- 测试集：')
print(test_data[['city', 'salary', 'city_target_train']])
print()
print('差异分析：')
print('全量均值：', all_mean.to_dict())
print('训练集均值：', train_mean.to_dict())
```

解析： 方式一使用全量数据计算均值，测试集的目标信息泄露到了编码中，导致模型评估过于乐观。方式二仅用训练集计算，是正确的做法。实际工作中建议使用交叉验证方式计算 Target Encoding，进一步减少泄露。

第六章 二进制编码（Binary Encoding）答案

✅ 练习 6-1：手动实现二进制编码

```
from sklearn.preprocessing import LabelEncoder
import pandas as pd
import numpy as np


def manual_binary_encode(series):
    """
    手动实现二进制编码
    1. 先用 LabelEncoder 获取整数编码
    2. 将整数转换为二进制位
    """
    # 步骤1：标签编码
    le = LabelEncoder()
    labels = le.fit_transform(series)


    # 步骤2：确定二进制位数
    n_categories = len(le.classes_)
    n_bits = int(np.ceil(np.log2(max(n_categories, 2))))


    # 步骤3：转换为二进制
    binary_cols = []
    for bit in range(n_bits):
        col_name = f'{series.name}_bit{bit}'
        binary_cols.append(
            pd.Series((labels >> bit) & 1, name=col_name)
        )


    return pd.DataFrame(binary_cols).T


# 应用到 city 列
city_binary_manual = manual_binary_encode(data['city'])
print('手动二进制编码结果：')
print(city_binary_manual)
print(f'\n类别数: {data["city"].nunique()}')
print(f'二进制位数: {city_binary_manual.shape[1]}')
```

预期输出：

```
手动二进制编码结果：
   city_bit0  city_bit1
0          0          0    # 北京 (0 -> 00)
1          1          0    # 上海 (1 -> 01)
2          0          1    # 广州 (2 -> 10)
```

```
3          0          0    # 北京
4          1          1    # 深圳 (3 -> 11)
...

类别数: 4
二进制位数: 2    # log2(4) = 2
```

✅ 练习 6-2：维度对比

```
import pandas as pd
import numpy as np
import category_encoders as ce
from sklearn.preprocessing import OneHotEncoder

# 创建包含 20 个类别的变量
categories = [f'cat_{i}' for i in range(20)]
test_series = pd.DataFrame({
    'feature': np.random.choice(categories, 100)
})

# One-Hot Encoding
ohe = OneHotEncoder(sparse_output=False)
ohe_result = ohe.fit_transform(test_series[['feature']])
ohe_dims = ohe_result.shape[1]

# Binary Encoding
be = ce.BinaryEncoder(cols=['feature'])
be_result = be.fit_transform(test_series[['feature']])
be_dims = be_result.shape[1]

print(f'One-Hot 编码维度: {ohe_dims}')
print(f'Binary  编码维度: {be_dims}')
print(f'维度减少比例: {(ohe_dims - be_dims) / ohe_dims * 100:.1f}%')
print(f'理论 Binary 维度: ceil(log2(20)) = {int(np.ceil(np.log2(20)))}')
```

预期输出：

```
One-Hot 编码维度: 20
Binary  编码维度: 5     # ceil(log2(20)) = 5
维度减少比例: 75.0%
理论 Binary 维度: 5
```

> **解析：** 20 个类别时，One-Hot 需要 20 列，Binary 只需 5 列 (ceil(log2(20))=5)，

维度减少了 75%。类别数越多，Binary Encoding 的优势越明显。

第七章 频率编码（Frequency Encoding）答案

✅ 练习 7-1：对多列进行频率编码

```
import pandas as pd


def freq_encode(df, columns):
    """
    对指定的多个列进行频率编码


    Parameters:
    -----------
    df : pd.DataFrame - 输入数据框
    columns : list - 需要编码的列名列表


    Returns:
    --------
    pd.DataFrame - 添加了频率编码列的数据框
    """
    df_encoded = df.copy()
    for col in columns:
        freq = df[col].value_counts(normalize=True)
        df_encoded[f'{col}_freq'] = df[col].map(freq)
    return df_encoded


# 应用到 city 和 color 列
result = freq_encode(data, ['city', 'color'])
print(result[['city', 'city_freq', 'color', 'color_freq']])
```

预期输出：

```
  city  city_freq color  color_freq
0  北京        0.3    红    0.400000
1  上海        0.3    蓝    0.300000
2  广州        0.2    绿    0.300000
3  北京        0.3    红    0.400000
4  深圳        0.2    蓝    0.300000
...
```

✅ 练习 7-2：频率冲突问题

```
import pandas as pd
import numpy as np


# 创建有频率冲突的数据
```

```
conflict_data = pd.DataFrame({
    'brand': ['Apple', 'Apple', 'Samsung', 'Samsung',
              'Huawei', 'Huawei', 'Xiaomi', 'Xiaomi',
              'Sony', 'Sony'],  # Apple 和 Samsung 频率相同
    'sales': [100, 120, 90, 110, 80, 85, 70, 75, 60, 65]
})

# 基本频率编码（有冲突）
freq = conflict_data['brand'].value_counts(normalize=True)
conflict_data['brand_freq'] = conflict_data['brand'].map(freq)

print('基本频率编码（有冲突）：')
print(conflict_data[['brand', 'brand_freq']].drop_duplicates())
print('\n问题：Apple 和 Samsung 编码相同！')

# 解决方案一：加入微小扰动
np.random.seed(42)
noise = np.random.uniform(-0.001, 0.001, len(conflict_data))
conflict_data['brand_freq_noise'] = conflict_data['brand_freq'] + noise

print('\n方案一 - 加入微小扰动：')
print(conflict_data[['brand', 'brand_freq_noise']].drop_duplicates())

# 解决方案二：结合频率+标签编码
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
conflict_data['brand_label'] = le.fit_transform(conflict_data['brand'])
conflict_data['brand_combined'] = (
    conflict_data['brand_freq'] * 100 + conflict_data['brand_label']
)

print('\n方案二 - 频率+标签组合：')
print(conflict_data[['brand', 'brand_combined']].drop_duplicates())
```

解析： 频率冲突在实际数据中很常见。加入微小扰动是最简单的解决方案，但会引入随机性。更稳定的方法是结合频率与其他编码信息，或者同时使用计数和频率作为两个特征。

第八章 综合实战练习答案

☑ 挑战 A：完整的数据编码流程

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder
import category_encoders as ce


# 创建数据
np.random.seed(42)
challenge_data = pd.DataFrame({
    'gender': ['男', '女', '男', '女', '男'] * 20,
    'blood_type': ['A', 'B', 'O', 'AB', 'A'] * 20,
    'job_level': ['初级', '中级', '高级', '专家', '初级'] * 20,
    'province': [f'省份_{i}' for i in range(100)],
    'target': np.random.randint(0, 2, 100)
})


print('原始数据概览：')
print(f'gender:    {challenge_data["gender"].nunique()} 个类别 - 无序、低基数')
print(f'blood_type:{challenge_data["blood_type"].nunique()} 个类别 - 无序、低基
数')
print(f'job_level: {challenge_data["job_level"].nunique()} 个类别 - 有序')
print(f'province:  {challenge_data["province"].nunique()} 个类别 - 无序、高基数
')


# ===== 1. gender: One-Hot Encoding (无序、仅 2 个类别) =====
ohe_gender = OneHotEncoder(sparse_output=False, drop='first')
gender_encoded = ohe_gender.fit_transform(challenge_data[['gender']])
challenge_data['gender_encoded'] = gender_encoded
print('\n1. gender -> One-Hot (drop first):')
print(f'   编码维度: {gender_encoded.shape[1]}')


# ===== 2. blood_type: One-Hot Encoding (无序、低基数) =====
ohe_blood = OneHotEncoder(sparse_output=False, drop='first')
blood_encoded = ohe_blood.fit_transform(challenge_data[['blood_type']])
print('\n2. blood_type -> One-Hot (drop first):')
print(f'   特征名: {ohe_blood.get_feature_names_out()}')
print(f'   编码维度: {blood_encoded.shape[1]}')


# ===== 3. job_level: Ordinal Encoding (有自然顺序) =====
level_order = [['初级', '中级', '高级', '专家']]
oe = OrdinalEncoder(categories=level_order)
challenge_data['job_encoded'] = oe.fit_transform(
    challenge_data[['job_level']]
)
print('\n3. job_level -> Ordinal:')
```

```
print('   映射: 初级->0, 中级->1, 高级->2, 专家->3')


# ===== 4. province: Target/Binary Encoding (高基数) =====
# 方案A: Target Encoding
te = ce.TargetEncoder(cols=['province'])
challenge_data['province_target'] = te.fit_transform(
    challenge_data['province'], challenge_data['target']
)


# 方案B: Binary Encoding
be = ce.BinaryEncoder(cols=['province'])
province_binary = be.fit_transform(challenge_data[['province']])


print('\n4. province:')
print(f'   Target Encoding 维度: 1')
print(f'   Binary Encoding 维度: {province_binary.shape[1]}')


print('\n编码选择总结：')
print('  gender     -> One-Hot (drop=first) : 无序、仅2类')
print('  blood_type -> One-Hot (drop=first) : 无序、仅4类')
print('  job_level  -> Ordinal              : 有序')
print('  province   -> Target / Binary      : 无序、100类高基数')
```

解析：编码选择的核心逻辑：① 先看是否有序：有序用 Ordinal；② 无序的看基数：低基数（<10）用 One-Hot，中基数（10-100）用 Binary，高基数（>100）用 Target；③ 同时考虑下游模型类型。

挑战 B：编码效果对比实验

```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
import category_encoders as ce


# 准备数据
np.random.seed(42)
n = 200
exp_data = pd.DataFrame({
    'city': np.random.choice(
        ['北京', '上海', '广州', '深圳', '成都',
         '武汉', '杭州', '南京'], n),
```

```
    'target': np.random.randint(0, 2, n)
})


# 方法 1: Label Encoding
le = LabelEncoder()
X_label = le.fit_transform(exp_data['city']).reshape(-1, 1)


# 方法 2: One-Hot Encoding
ohe = OneHotEncoder(sparse_output=False)
X_onehot = ohe.fit_transform(exp_data[['city']])


# 方法 3: Target Encoding
te = ce.TargetEncoder(cols=['city'])
X_target = te.fit_transform(
    exp_data[['city']], exp_data['target']
).values


# 训练并比较
y = exp_data['target']
dt = DecisionTreeClassifier(random_state=42, max_depth=3)


methods = {
    'Label Encoding': X_label,
    'One-Hot Encoding': X_onehot,
    'Target Encoding': X_target
}


print('编码方法对比实验结果：')
print('=' * 50)
for name, X in methods.items():
    scores = cross_val_score(dt, X, y, cv=5, scoring='accuracy')
    print(f'{name:20s}: '
          f'均值={scores.mean():.4f}, '
          f'标准差={scores.std():.4f}, '
          f'维度={X.shape[1]}')
```

预期输出示例：

```
编码方法对比实验结果：
==================================================
Label Encoding      : 均值=0.4850, 标准差=0.0412, 维度=1
One-Hot Encoding    : 均值=0.5100, 标准差=0.0367, 维度=8
Target Encoding     : 均值=0.5250, 标准差=0.0523, 维度=1


注意：由于数据是随机生成的，实际准确率可能与上述不同。
重要的是理解实验设计和对比方法。
```

解析：本实验使用随机数据，所以编码方法之间的差异可能不明显。在真实数据中，合适的编码方法可以显著提升模型效果。核心要点是学会使用 cross_val_score 进行公平对比，并关注均值和标准差两个指标。

第九章 自我检测答案

9.1 知识检测答案

（✗）Label Encoding 适合用于线性回归模型中的无序分类变量

Label Encoding 会引入虚假的大小关系，线性模型会利用这种关系导致错误结果。应该使用 One-Hot Encoding。

（✓）One-Hot Encoding 会引入多重共线性问题，可以通过 drop='first' 解决

正确。n 个类别的 One-Hot 列之和总为1，存在完全共线性。删除一列可解决这个问题且不丢失信息。

（✓）Ordinal Encoding 要求变量必须具有自然顺序

正确。Ordinal Encoding 的前提是变量有明确的顺序关系，否则编码出的数值顺序没有意义。

（✗）Target Encoding 的主要风险是维度爆炸

Target Encoding 的主要风险是目标泄露（target leakage），不是维度爆炸。它只生成 1 列，不增加维度。

（✓）Binary Encoding 的输出维度为 log2(n)

正确。更精确地说是 ceil(log2(n))，即向上取整。例如 5 个类别需要 ceil(log2(5)) = 3 位。

（✗）Frequency Encoding 不会出现不同类别编码相同的问题

错误。当不同类别出现频率相同时，它们的编码值就会相同，这就是频率冲突问题。